МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

федеральное государственное бюджетное образовательное учреждение высшего образования

«ИРКУТСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

ФГБОУ ВО «ИГУ»

Институт математики и информационных технологий

Кафедра алгебраических и
информационных систем

ОТЧЕТ ПО УЧЕБНОЙ ПРАКТИКЕ

Руководитель:
профессор кафедры алгебраических и
информационных систем, д.ф.-м.н.
Винокуров С.Ф.

Студенты:
Петров Иван Алексеевич,
Дондукова Виктория Евгеньевна,
гр. 02361-ДБ, 02362-ДБ 3 курс ПИ

Иркутск 2024

Дневник прохождения учебной практики

Дата	**Краткое содержание работы**	**Отметка о выполнении**
27.05.24	Получение задания на практику.	
с 27.05.24 по 29.05.24	Изучение и анализ алгоритма муравьиной колонии.	
с 29.05.24 по 31.05.24	Перенос алгоритма на задачу.	
с 31.05.24 по 01.06.24	Разработка структуры программы.	
с 01.06.24 по 02.06.24	Тестирование и отладка программы.	
с 02.06.24 по 03.06.24	Исследования настройки параметров алгоритма муравьиной колонии.	
с 03.06.24 по 05.06.24	Разработка графического интерфейса пользователя.	
с 05.06.24 по 08.06.24	Оформление отчета по производственной практике.	

Содержание

Введение

Алгоритм муравьиной колонии (ACO) — это метаэвристический метод оптимизации, вдохновленный поведением муравьев при поиске пищи. Он применяется для решения различных задач оптимизации, таких как задачи нахождения кратчайшего пути, задачи о рюкзаке и другие комбинаторные задачи. Основной идеей является использование коллективного поведения муравьев для поиска оптимальных решений, что похоже на то, как настоящие муравьи находят наилучший путь от гнезда к источнику пищи.

В ACO муравьи представляют собой агенты, которые исследуют пространство поиска, оставляя за собой феромоны — химические следы, которые влияют на выбор пути других муравьев. Сначала каждый муравей случайным образом выбирает путь, но в дальнейшем его выбор будет зависеть от концентрации феромонов на различных путях. Чем больше феромона на пути, тем выше вероятность, что другой муравей выберет этот же путь. Такой механизм позволяет системе коллективно двигаться к лучшим решениям.

Алгоритм состоит из нескольких этапов. На начальном этапе создаются "популяции" муравьев, каждый из которых начинает искать решение, оставляя феромоны на пути. По мере выполнения алгоритма феромоны испаряются, что способствует уменьшению влияния менее эффективных путей. Муравьи обновляют свои траектории, основываясь на полученных данных о концентрации феромонов, а также на своем опыте.

Алгоритм использует функцию оценки (fitness-функцию), чтобы измерить качество найденных решений. В контексте задачи о выполнимости КНФ, муравьи представляют собой агенты, которые ищут такую установку значений переменных (истина/ложь), при которой все дизъюнкции в КНФ-формуле будут истинными. Идея заключается в том, чтобы муравьи исследовали пространство возможных установок, выбирая комбинации значений для переменных, оставляя за собой феромоны, которые будут влиять

на выбор других муравьев, и таким образом приводить к нахождению удовлетворяющего решения.

Как и другие методы метаэвристики, ACO не гарантирует нахождение глобального оптимума, но способен найти достаточно хорошие решения за разумное время, что делает его эффективным для решения сложных задач, где традиционные методы могут быть слишком затратными или трудными для реализации.

Раздел 1. Исследование предметной области

1.1. Описание предметной области

Задача о выполнимости КНФ (конъюнктивной нормальной формы) является одной из классических задач теории вычислительных наук и теории сложности. В общем виде задача заключается в том, чтобы определить, существует ли такая установка значений переменных, при которой логическое выражение, представленное в виде КНФ, будет истинным. КНФ представляет собой логическое выражение, которое состоит из нескольких дизъюнкций (или) литералов, объединенных с помощью конъюнкций (и).

Математически задача о выполнимости КНФ формулируется следующим образом: пусть имеется логическое выражение в виде КНФ, состоящее из m клауз (дизъюнкций) и n переменных. Каждая клауза C_j — это дизъюнкция литералов, которые могут быть как положительными, так и отрицательными. Литералы — это переменные или их отрицания. Задача заключается в том, чтобы найти такую установку значений для всех переменных, при которой каждая из клауз будет истинной.

Математически, для данного выражения в КНФ, необходимо определить, существует ли такая установка переменных $x_1, x_2, \dots, x_n$ (где $x_i \in \{0, 1\}$, при которой все клауз $C_1, C_2, \dots, C_m$ будут истинными. Каждая клауза C_j считается истинной, если хотя бы один из её литералов имеет значение 1. В противном случае клауза ложна.

Задача о выполнимости КНФ является NP-полной, что означает, что для неё не существует полиномиального алгоритма, который решает задачу за разумное время в общем случае. В связи с этим, для решения данной задачи часто применяются методы метаэвристики, такие как генетические алгоритмы, алгоритмы муравьиной колонии или алгоритм имитации отжига.

В данной работе была рассмотрена задача о выполнимости КНФ и применен метод решения данной задачи с использованием алгоритма муравьиной колонии. Алгоритм реализован на языке Java с использованием библиотеки JavaFX.

1.2. Постановка задачи

Целью данной работы является разработка и реализация алгоритма муравьиной колонии (ACO) для решения задачи о выполнимости КНФ (конъюнктивной нормальной формы). В рамках работы будут выполнены следующие задачи:

- Разработка и реализация ACO для данной задачи.
- Создание графического интерфейса пользователя для запуска и визуализации работы алгоритма.
- Проведение тестирования и анализа эффективности разработанного решения.

1.3. Описание алгоритма

Алгоритм муравьиной колонии (ACO) вдохновлен поведением муравьев в природе. Муравьи при поиске пищи используют феромоны, чтобы обмениваться информацией с другими особями и эффективно находить кратчайшие пути. Это поведение легло в основу алгоритма, где каждый муравей представляет собой поиск решения задачи, а феромоны служат индикатором для принятия решений другими муравьями.

В контексте задачи о выполнимости КНФ, задача сводится к поиску такого набора значений для переменных, при котором все клаузы логического выражения будут истинными. Каждый муравей строит решение, выбирая для каждой переменной значение 0 или 1 на основе информации о феромонах, оставленных другими муравьями. После завершения построения решения оценивается его качество с помощью функции пригодности, которая проверяет, является ли выражение в КНФ выполненным для данного набора значений.

Цель алгоритма — найти такое решение, при котором каждая из клауз КНФ будет истинной, то есть хотя бы один литерал в каждой клаузе должен быть истинным. Решения строятся итерационно, с каждым шагом улучшая свое положение, следуя за решениями, которые приводят к лучшим результатам.

На каждой итерации феромоны обновляются по следующему правилу:

$$\tau_{ij}(t+1) = (1-p) * \tau_{ij}(t) + \Delta_{\tau_{ij}}$$

где $\tau_{ij}(t+1)$ — — уровень феромонов на пути от вершины i к вершине j в момент времени t, p — коэффициент испарения феромонов, $\Delta_{\tau_{ij}}$ количество феромонов, добавленных на этот путь, которое зависит от качества найденного решения..

Каждый муравей строит решение, выбирая значение для каждой переменной на основе вероятности, пропорциональной количеству феромонов на пути, а также случайности:

$$P(x_i = 1) = \frac{\tau_{i1}}{\sum_{j=1}^{2} \tau_{ij}}$$

где $P(x_i = 1)$ — вероятность того, что переменная x_i примет значение 1, а τ_{i1} — уровень феромонов на пути, ведущем к значению 1 для переменной x_i.

Раздел 2. Описание реализации проекта

2.1. Функции, используемые в работе алгоритма

Вот краткое описание функций, связанных с алгоритмом:

reading(String s)

1. Проходит по строке s, ищет скобки, а также символы, представляющие логические переменные и операторы;
2. Для каждой переменной (например, x1, ¬x1, x2 и т.д.) добавляет информацию в структуру данных expression и variable;
3. Связывает переменные и выражения, которые с ними ассоциируются, формируя структуру для дальнейшей работы с алгоритмом муравьиной колонии.

sort_variable()

1. Использует коллекцию variable, содержащую информацию о переменных и их связях с выражениями;
2. Сортирует переменные, делая те, которые чаще встречаются, более приоритетными в алгоритме муравьиной колонии.

sort_expression()

1. Использует коллекцию expression, которая содержит информацию о выражениях и связанных с ними переменных;
2. Сортирует выражения, делая те, которые содержат больше переменных, более приоритетными для решения.

phero_random_variable(HashMap<Integer, Double> Jik)

1. Сначала вычисляется сумма всех феромонов из карты Jik;
2. Затем генерируется случайное число, которое позволяет выбрать переменную, пропорциональную её феромонному следу;
3. Возвращает выбранную переменную.

ACO()

1. Инициализирует параметры алгоритма, такие как количество итераций, коэффициенты для феромонов, минимальное значение феромона и т.д.;
2. Проходит через несколько итераций, моделируя поведение муравьёв, которые решают задачу;
3. Для каждого муравья генерируется решение, и обновляется феромон на основе того, как муравей прошёл по переменным;
4. В процессе итерации муравьи выбирают переменные и модифицируют их значения, основываясь на феромонных путях;
5. После каждого завершения итерации обновляется феромон, и выбираются лучшие решения, которые сохраняются как наилучшие;
6. Программа также выводит промежуточные и конечные результаты решения задачи, такие как количество переменных, которые муравьи смогли разрешить в КНФ, и их значения.

2.2. Тестовые примеры

Для апробации алгоритма сгенерированы тестовые входные данные для программы. Для обоих примеров использовались следующие параметры: кол-во итераций: 100, коэффициент феромона: 1.0, минимальный феромон: 0.1, изменение феромона при пробеге: 0.05, изменение феромона при неактивной переменной: 0.02.

2.2.1. Первый пример

(x1 + x2) (x3 + !x4 + x5) (x6 + !x7 + x8) (x9 + x10 + !x11 + x12) (x13 + x14) (x15 + !x16 + x17) (x18 + !x19 + x20) (x21 + !x22 + x23) (x24 + x25 + !x1)

Результат: Наилучшая найденная комбинация = {1, (1 || 0), (1 || 0), (1 || 0), 1, 1, 0, 1, 1, 1, 0, 1, 1, 1, 1, 0, 1, 1, 0, 1, 1, 0, 1, 1, 1.

2.2.2. Второй пример

В качестве второго примера посмотрим результат алгоритма при неразрешимой КНФ: (x1 + x2) (x1 + !x2) (!x1 + x2) (!x1 + !x2)

Результат: Алгоритм не смог найти решение для данной КНФ.

2.3. Результаты

Для исследования работы описанного ранее алгоритма и его работы в разработанном приложении (рис. 1) составлены следующие графики:

- Влияние параметра min_phero (минимальное значение феромона) на работу алгоритма – рис. 2-4;
- Влияние параметра mod_phero (увеличение феромона при прохождении муравья) на работу алгоритма – рис. 5-7;
- Влияние параметра minus_phero (уменьшение феромона при отсутствии прохождения муравья) на работу алгоритма – рис. 11-13.

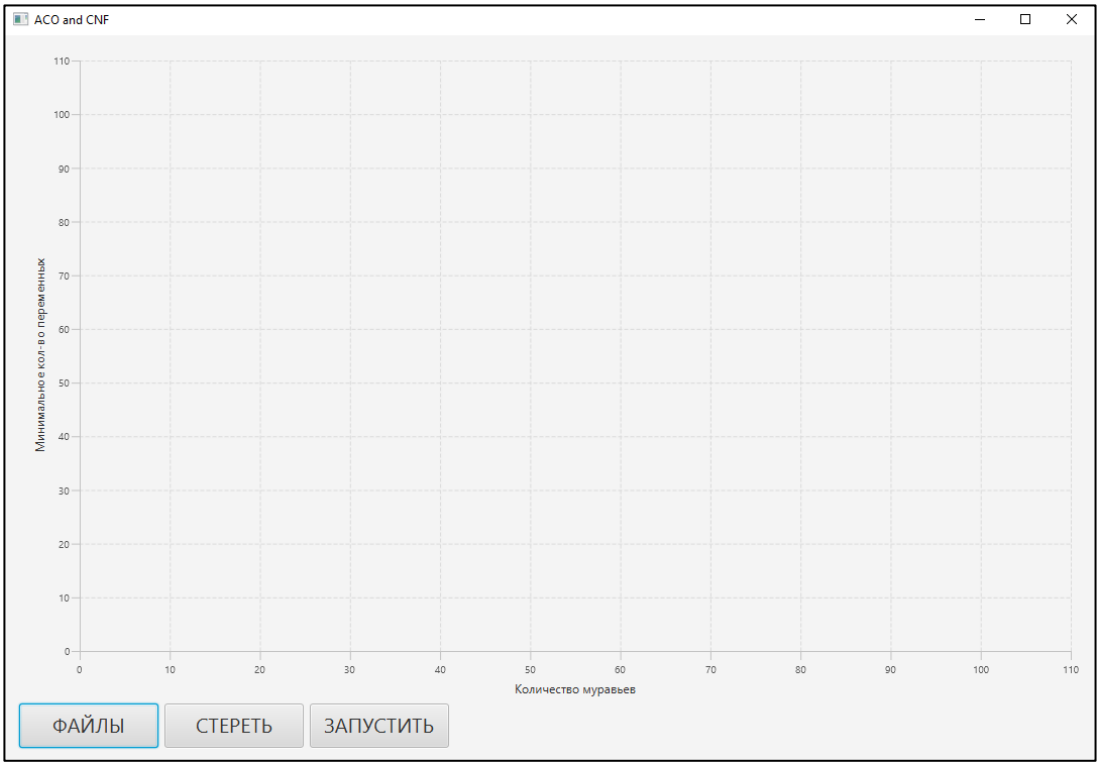

Рис. 1 Внешний вид приложения

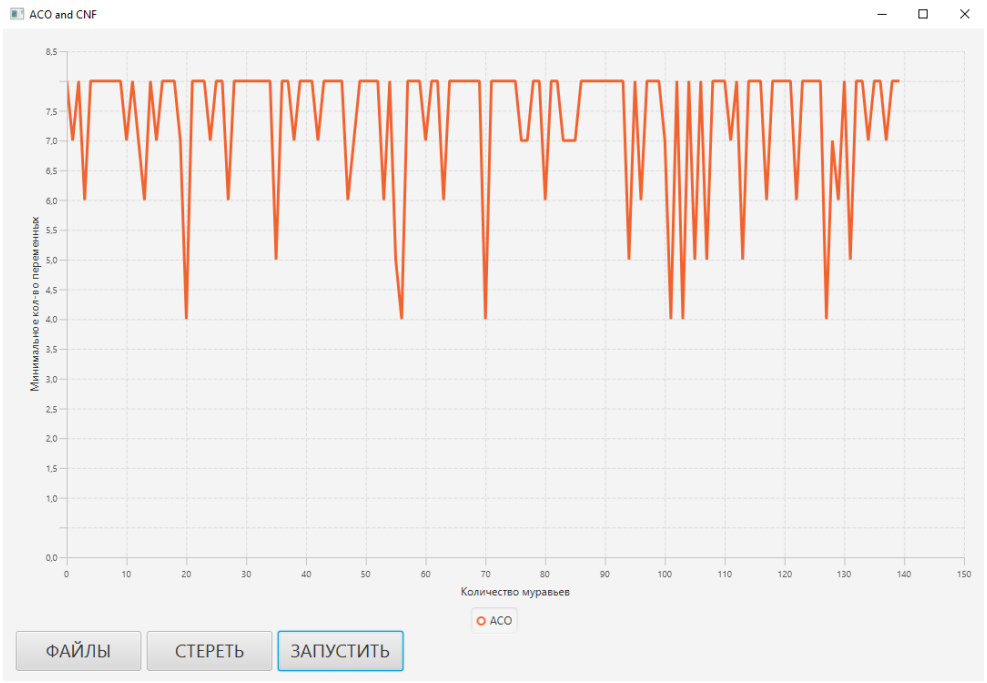

Рис. 2 Вычисления алгоритма при min_phero = 0.1

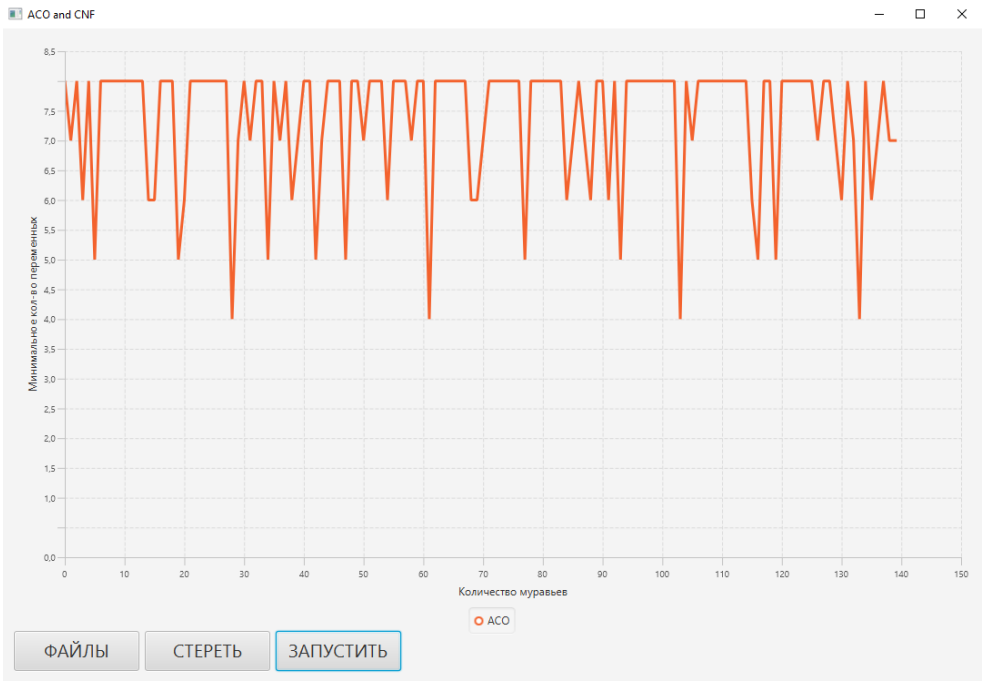

Рис. 3 Вычисления алгоритма при min_phero=0.5

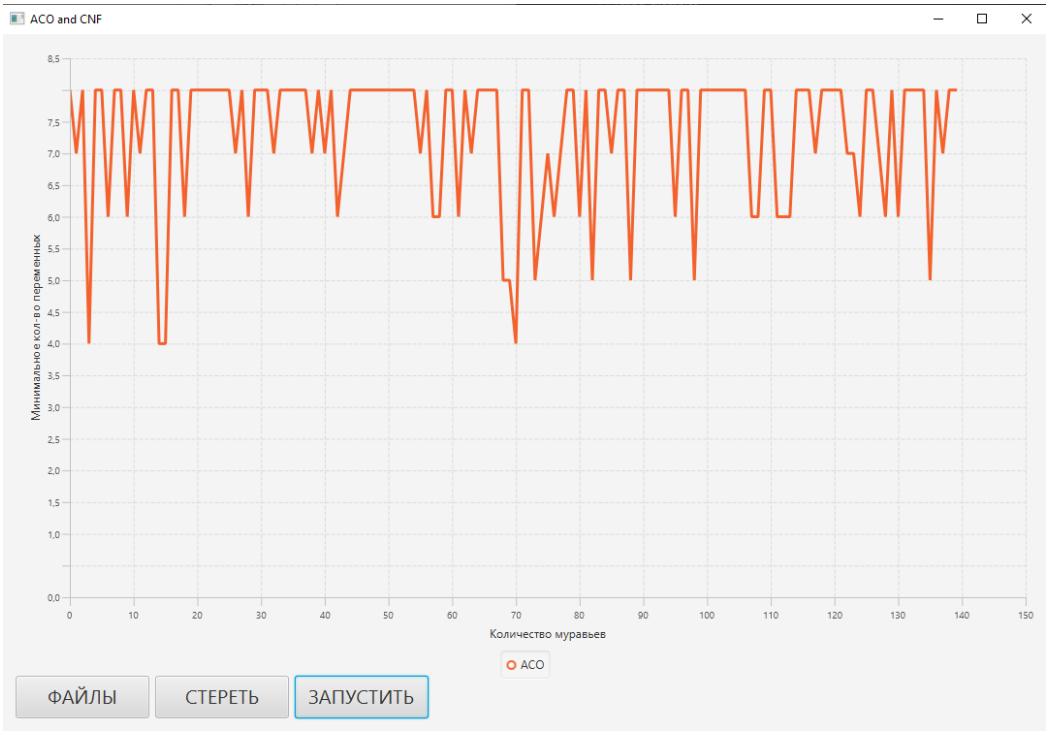

Рис. 4 Вычисления алгоритма при min_phero=1.0

Из графиков на рис. 2, 3 и 4 можно сделать вывод, что минимальное количество феромона почти не влияет на скорость поиска решения.

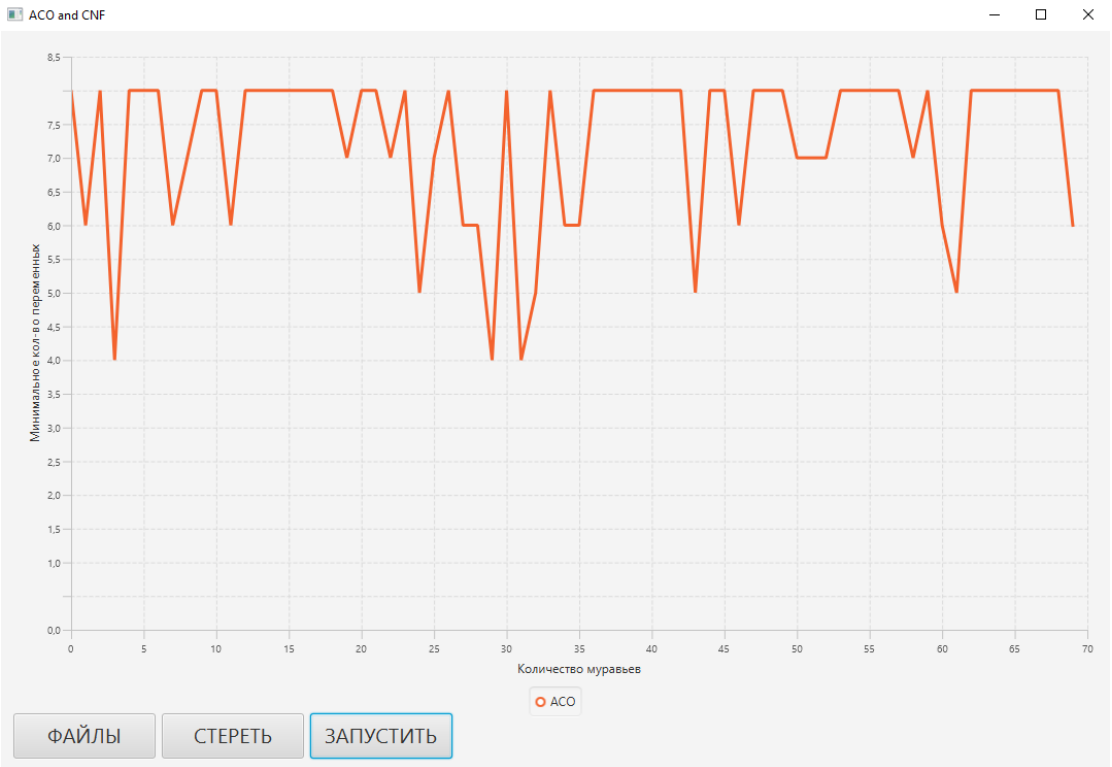

Рис. 5 Вычисления алгоритма при mod_phero=0.05

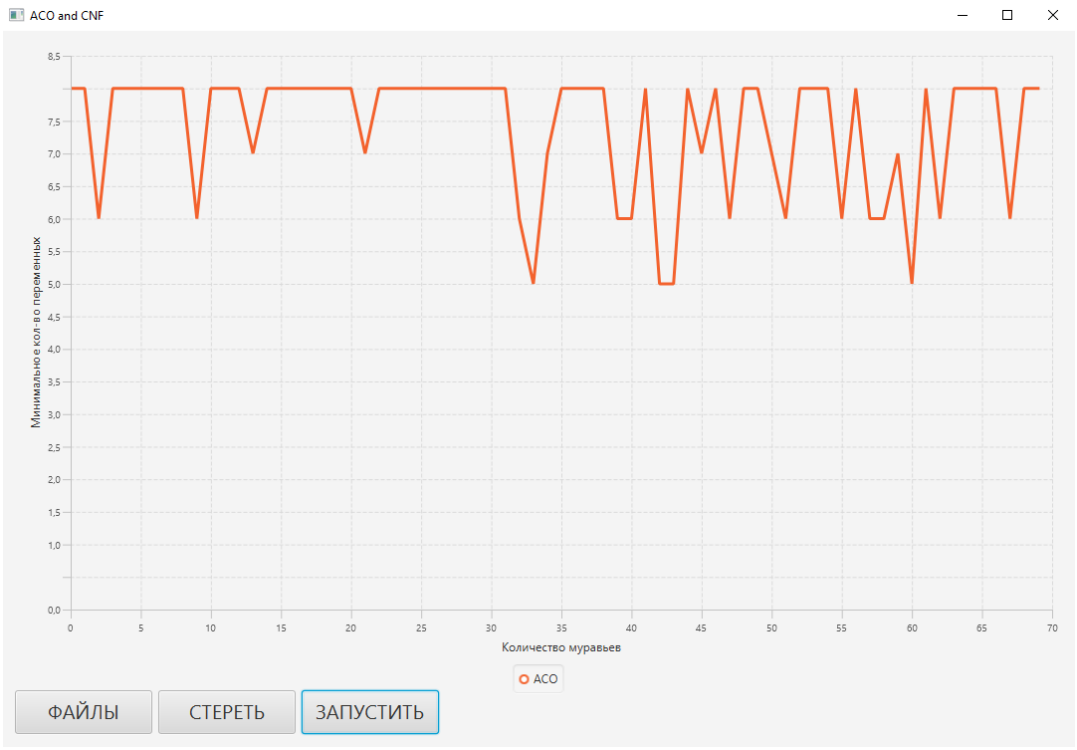

Рис. 6 Вычисления алгоритма при mod_phero=0.5

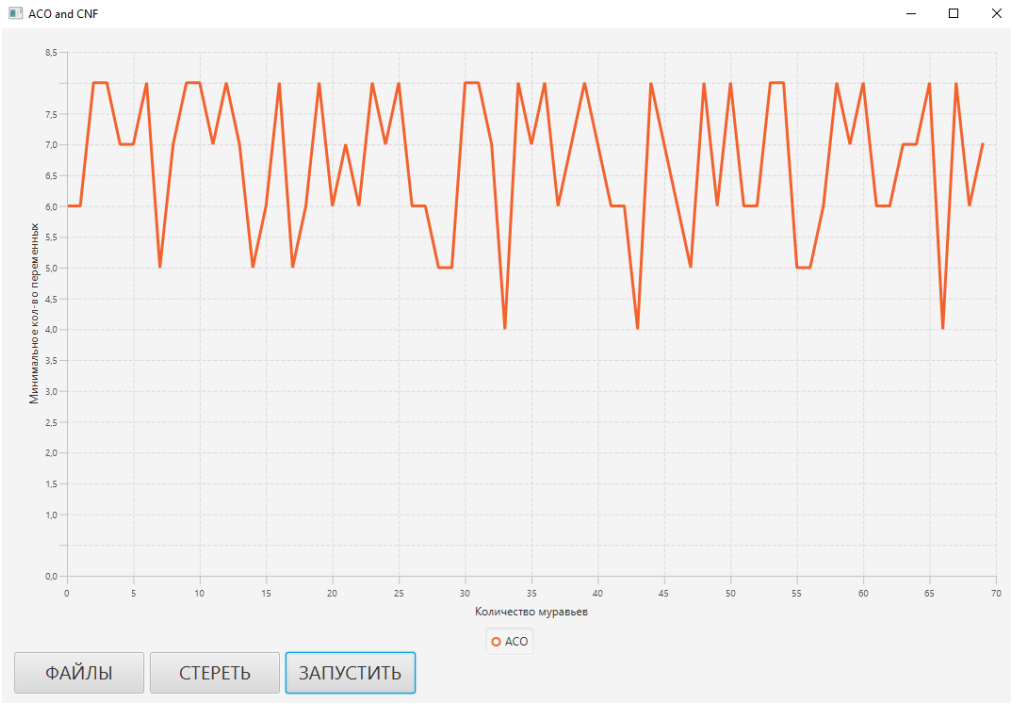

Рис. 7 Вычисления алгоритма при mode_phero=0.9

Из анализа графиков на рисунках 5, 6 и 7 следует, что увеличение значения феромона оказывает существенное влияние на работу алгоритма, ускоряя процесс нахождения оптимального решения.

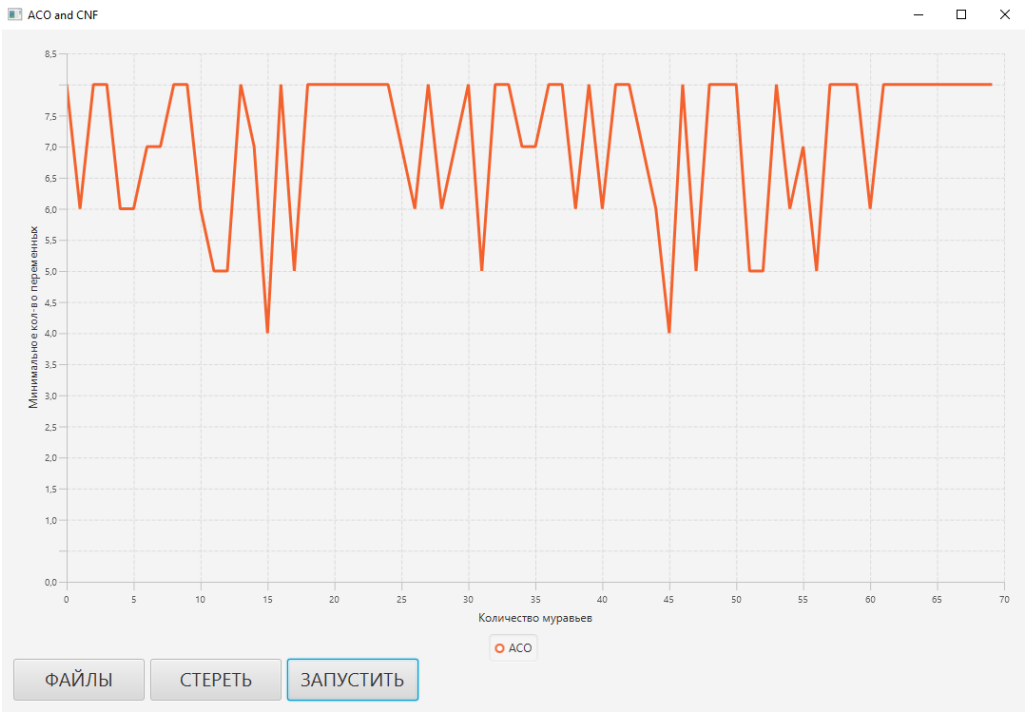

Рис.8 Вычисления алгоритма при minus_phero=0.02

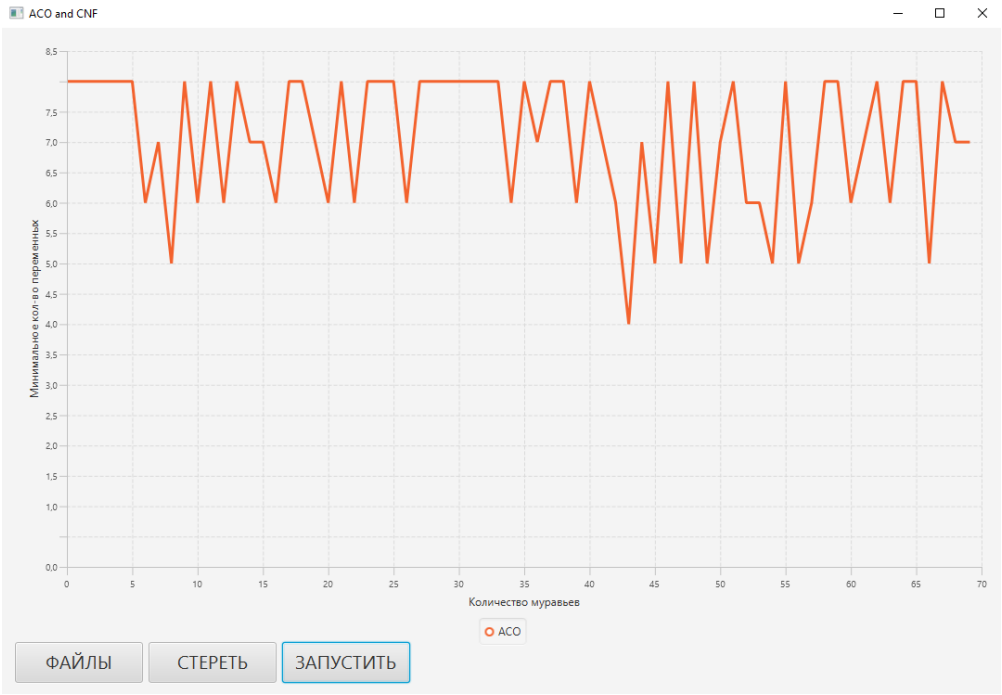

Рис.9 Вычисления алгоритма при minus_phero=0.5

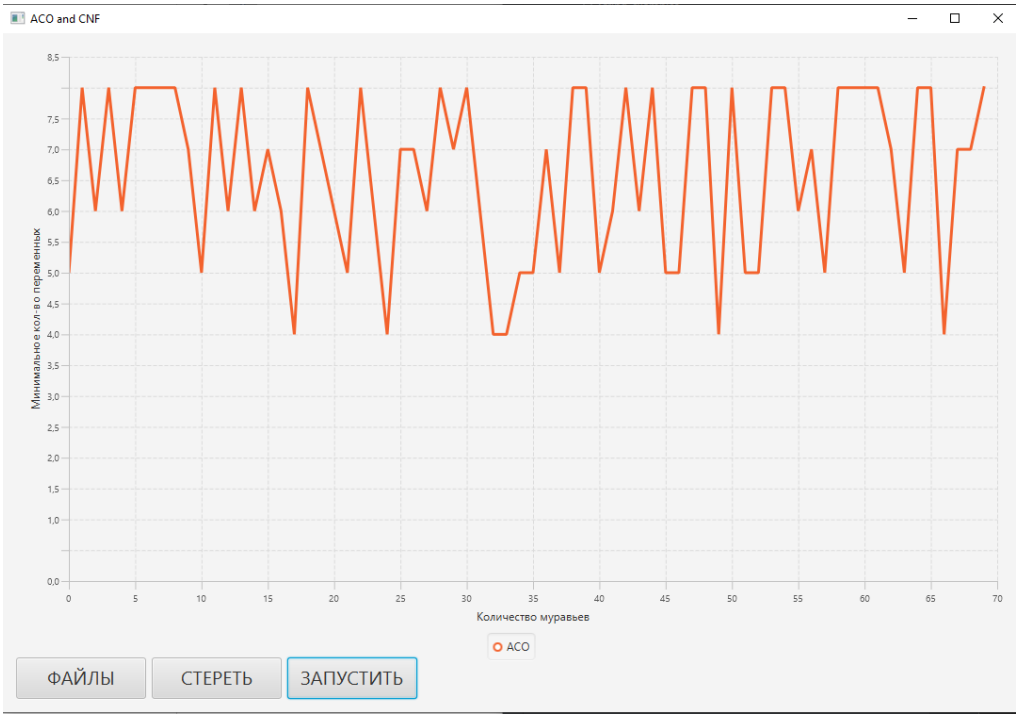

Рис.10 Вычисления алгоритма при minus_phero=1.0

Анализ графиков на рисунках 8-10 позволяет сделать вывод, что более быстрое снижение феромона при отсутствии прохождения муравья через вершину способствует увеличению числа локальных минимумов.

Из вышеизложенного анализа можно сделать вывод, что изменение каждого из параметров оказывает значительное влияние на эффективность работы алгоритма. Однако, важно подчеркнуть, что корректировка этих параметров по отдельности может привести к нежелательным последствиям, таким как замедление поиска решения или увеличение числа локальных минимумов. Поэтому такие изменения должны проводиться тщательно и обдуманно, с учётом взаимодействия всех параметров, для достижения оптимальных результатов.

Заключение

В ходе исследования был реализован и протестирован алгоритм муравьиной колонии для проверки выполнимости КНФ. Проведённые вычислительные эксперименты показали, что алгоритм стабильно и быстро находит решение.

Анализ результатов продемонстрировал, что варьирование параметров алгоритма не оказывает значительного влияния на конечный результат. Независимо от значений параметров, алгоритм эффективно решает задачу, находя выполняющие присваивания переменным, если таковые существуют.

Таким образом, муравьиный алгоритм показал свою эффективность для решения задачи SAT, подтверждая свою способность быстро находить удовлетворяющие присваивания в логических формулах.

Список литературы

1. Браунли, Дж. Умные алгоритмы: Программирование, вдохновленное природой / Дж. Браунли. — 2011. — ISBN: 978-1-4467-8506-5.
2. Светлов А. А. "Муравьиные алгоритмы" // ResearchGate. [Электронный ресурс] – URL: https://researchgate.net/publication/280064579_Muravinye_algoritmy. Дата обращения: 27.05.2024.
3. Бедакова Н. В. "Алгоритм муравьиной колонии для решения задач оптимального размещения распределительных центров розничной торговой сети" // CyberLeninka. [Электронный ресурс] – URL: https://cyberleninka.ru/article/n/algoritm-muravinoy-kolonii-dlya-resheniya-zadach-optimalnogo-razmescheniya-raspredelitelnyh-tsentrov-roznichnoy-torgovoy-seti. Дата обращения 29.05.2024.

Приложение

Листинг 1 – Класс «HelloController» представляет собой контроллер для решения задачи о выполнимости КНФ с использованием алгоритма поиска на основе муравьиной колонии (Ant Colony Optimization, ACO). Он осуществляет работу с входными данными, выполняет алгоритм ACO и визуализирует результаты.

```
import javafx.event.ActionEvent;
import javafx.fxml.FXML;

import java.io.File;
import java.io.FileNotFoundException;
import java.io.IOException;
import java.util.*;

public class HelloController {

String currentWorkingDirectory = System.getProperty("user.dir");
File parameter_file = new File(currentWorkingDirectory +
"/resources/txt.txt");
File knf_file = new File(currentWorkingDirectory + "/resources/knf.txt");
ArrayList<Trio <Integer, Integer, ArrayList<Integer>>> expression = new
ArrayList<Trio<Integer, Integer, ArrayList<Integer>>>();
ArrayList<Trio <Integer, Integer, ArrayList<Integer>>> variable = new
ArrayList<Trio<Integer, Integer, ArrayList<Integer>>>();
HashMap<Integer, Integer>  all_variables = new HashMap<>();

public void Reading(String s){
int var = 0;

for(int i = 0 ; i < s.length(); i++){
if(s.charAt(i) == '('){
this.expression.add(new Trio<>(var, 0, new ArrayList<>()));
ArrayList<Integer> list = new ArrayList<>();
while(s.charAt(i) != ')'){
    if(s.charAt(i) == '¬' || s.charAt(i) == '!' || s.charAt(i) == '-'){
        i += 2;
        StringBuffer buf = new StringBuffer();
        while(s.charAt(i) >= '0' && s.charAt(i) <= '9'){
            buf.append(s.charAt(i));
            i++;
        }
        int i1 = Integer.parseInt(buf.toString()) * -1;
        list.add(i1);

        if(!this.all_variables.containsKey(i1 * -1)){
            all_variables.put(i1 * -1, -1);
        }
```

```
            i--;

            int j = 0;
            int j1 = 0;

            for(Trio T : variable){
                if((int) T.getKey() == i1){
                    j1 = 1;
                    break;
                }
                j++;
            }

            if(j1 == 0){
                this.variable.add(new Trio<>(i1, 1, new ArrayList<>()));
                this.variable.get(j).getValue2().add(var);
            }
            else{

    this.variable.get(j).setValue1(this.variable.get(j).getValue1() + 1);
                this.variable.get(j).getValue2().add(var);
            }
        }
        else if(s.charAt(i) == 'x' || s.charAt(i) == 'X'){
            i += 1;
            StringBuffer buf = new StringBuffer();
            while(s.charAt(i) >= '0' && s.charAt(i) <= '9'){
                buf.append(s.charAt(i));
                i++;
            }
            int i1 = Integer.parseInt(buf.toString());
            list.add(i1);

            if(!this.all_variables.containsKey(i1)){
                all_variables.put(i1, -1);
            }

            i--;

            int j = 0;
            int j1 = 0;

            for(Trio T : variable){
                if((int) T.getKey() == i1){
                    j1 = 1;
                    break;
                }
                j++;
            }

            if(j1 == 0){
```

```
            this.variable.add(new Trio<>(i1, 1, new ArrayList<>()));
            this.variable.get(j).getValue2().add(var);
        }
        else{

this.variable.get(j).setValue1(this.variable.get(j).getValue1() + 1);
            this.variable.get(j).getValue2().add(var);
        }
    }
    i++;
}

this.expression.get(var).setValue1(list.size());
this.expression.get(var).setValue2(new ArrayList<>(list));
var++;
}
}
}


public void sort_variable(){
Collections.sort(this.variable, (p1, p2) ->
p2.getValue1().compareTo(p1.getValue1()));
}


public void sort_expression(){
Collections.sort(this.expression, (p1, p2) ->
p2.getValue1().compareTo(p1.getValue1()));
}


public static int phero_random_variable(HashMap<Integer, Double> Jik) {
Random rand = new Random();
int result = -1;

double totalPheromone = 0.0;
for (double pheromone : Jik.values()) {
totalPheromone += pheromone;
}

double randValue = rand.nextDouble() * totalPheromone;
double cumulative = 0.0;

for (Map.Entry<Integer, Double> entry : Jik.entrySet()) {
cumulative += entry.getValue();
if (cumulative >= randValue) {
result = entry.getKey();
break;
}
}
```

```
System.out.println(result);
return result;
}



public void ACO() throws FileNotFoundException {
Scanner in = new Scanner(parameter_file);
int var = Integer.parseInt(in.nextLine());
double k_phero = Double.parseDouble(in.nextLine());
double min_phero = Double.parseDouble(in.nextLine());
double mod_phero = Double.parseDouble(in.nextLine());
double minus_phero = Double.parseDouble(in.nextLine());

HashMap<Integer, Double> phero = new HashMap<>();
HashMap<Integer, Double> phero_mod = new HashMap<>();
Pair<Integer, HashMap<Integer, Integer>> best = new
Pair<>(this.all_variables.size() + 1, new HashMap<>());

for(int i = 0; i < this.variable.size(); i++){
phero.put(this.variable.get(i).getKey(), k_phero);
phero_mod.put(this.variable.get(i).getKey(), 0.0);

}
int i1 = 0;
for(int i = 0; i < var; i++){
System.out.println("RUN: " + i);

for(int j = 0; j < this.variable.size(); j++){

HashMap<Integer, Integer> cont = new HashMap<>(this.all_variables);

HashMap<Integer, Double> Jik = new HashMap<>(phero);

ArrayList<Trio <Integer, Integer, ArrayList<Integer>>> local_expression =
new ArrayList<>(this.expression);
ArrayList<Trio <Integer, Integer, ArrayList<Integer>>> local_variable =
new ArrayList<>(this.variable);

System.out.println("");
System.out.println("NEW ANT");
System.out.println("ant variable: " + this.variable.get(j).getKey());
Jik.remove(this.variable.get(j).getKey());
Jik.remove(this.variable.get(j).getKey() * -1);

if(this.variable.get(j).getKey() < 0){
    cont.put(this.variable.get(j).getKey() * -1, 0);
}
else{
    cont.put(this.variable.get(j).getKey(), 1);
```

```
}

int targetValue = this.variable.get(j).getKey();

Iterator<Trio<Integer, Integer, ArrayList<Integer>>> iterator =
local_expression.iterator();
while (iterator.hasNext()) {
    Trio<Integer, Integer, ArrayList<Integer>> trio = iterator.next();

    if (trio.getValue2().contains(targetValue)) {
        iterator.remove();
    }
}

System.out.println("LOCAL_EXPRESSION");
for(Trio T : local_expression){
    System.out.println(T.getKey() + " " + T.getValue1() + " " +
T.getValue2());
}
boolean flag = false;
while(!Jik.isEmpty() && flag != true){
    targetValue = phero_random_variable(Jik);
    System.out.println("ant variable: " + targetValue);
    Jik.remove(targetValue);
    Jik.remove(targetValue * -1);

    if(targetValue < 0){
        cont.put(targetValue * -1, 0);
    }
    else{
        cont.put(targetValue, 1);
    }

    Iterator<Trio<Integer, Integer, ArrayList<Integer>>> iterator1 =
local_expression.iterator();
    while (iterator1.hasNext()) {
        Trio<Integer, Integer, ArrayList<Integer>> trio =
iterator1.next();

        if (trio.getValue2().contains(targetValue)) {
            iterator1.remove();
        }
    }

    System.out.println("LOCAL_EXPRESSION");
    for(Trio T : local_expression){
        System.out.println(T.getKey() + " " + T.getValue1() + " " +
T.getValue2());
    }

    if(local_expression.isEmpty()){
```

```
        for (Map.Entry<Integer, Integer> entry : cont.entrySet()) {
            System.out.println("var " + entry.getKey() + " = " +
entry.getValue());
        }
        System.out.println("KNF = 1");
        flag = true;

        int sum = 0;

        for(Integer key : cont.keySet()) {
            Integer value = cont.get(key);
            if(value != -1){
                sum++;
                if(value == 0) {
                    phero.put(key * -1, phero.get(key * -1) + mod_phero);
                }
                else {
                    phero.put(key, phero.get(key) + mod_phero);
                }
            }
            else {
                if(phero.containsKey(key)) {
                    phero.put(key, phero.get(key) - minus_phero);
                    if(phero.get(key) < min_phero) {
                        phero.put(key, min_phero);
                    }
                }

                if(phero.containsKey(key * -1)) {
                    phero.put(key * -1, phero.get(key * -1) -
minus_phero);
                    if(phero.get(key * -1) < min_phero) {
                        phero.put(key * -1, min_phero);
                    }
                }
            }
        }
        if(best.getKey() > sum) {
            best = new Pair<>(sum, new HashMap<>(cont));
        }
        HelloApplication.add(i1, sum);
        i1++;
    }
}
if(!flag) {
    HelloApplication.add(i1, all_variables.size() + 1);
    i1++;
}
}

}
```

```
HelloApplication.lineChart.getData().add(HelloApplication.series);
if(best.getKey() != all_variables.size() + 1) {
System.out.println("Best combination: ");
System.out.println("count variables: " + best.getKey());
System.out.println("Variables:");
for (Map.Entry<Integer, Integer> entry : best.getValue().entrySet()) {
if(entry.getValue() != -1){
    System.out.println("x" + entry.getKey() + " = " + entry.getValue());
}
else{System.out.println("x" + entry.getKey() + " = (1 || 0)");}

}
}
else{System.out.println("Solution not found.");}
}

@FXML
void RUN_BUTTON(ActionEvent event) throws IOException {
HelloApplication.series.getData().clear();

this.variable.clear();
this.expression.clear();

Scanner in = new Scanner(knf_file);
String s = in.nextLine();
Reading(s);
sort_variable();
sort_expression();

System.out.println("EXPRESSION");
for(Trio T : this.expression){
System.out.println(T.getKey() + " " + T.getValue1() + " " +
T.getValue2());
}

System.out.println("VARIABLE");
for(Trio T : this.variable){
System.out.println(T.getKey() + " " + T.getValue1() + " " +
T.getValue2());
}

System.out.println("ALL_VARIABLES");
for(Map.Entry<Integer, Integer> entry : this.all_variables.entrySet()){
System.out.println(entry.getKey() + " " + entry.getValue());
}
ACO();
}
}
```